Asia Mathematika
Volume: 7 Issue: 2 , (2023) Pages: 27 – 34
Available online at **www.asiamath.org**

Noise classification of ICF images using a convolutional neural network (CNN)

Naima Naheed[1*], **Bradley T. Wolfe** [2] **and Zhehui Wang**[3]
[1]Department of Mathematics, Benedict College,
Columbia, SC, USA. ORCID iD: 0000-0003-0003-3278
[2]Los Alamos National Laboratory
Los Alamos, NM, USA. ORCID iD: 0000-0002-6830-1614
[3]Los Alamos National Laboratory
Los Alamos, NM, USA. ORCID iD: 0000-0001-7826-4063

Received: 26 May 2023 • **Accepted:** 24 Jul 2023 • **Published Online:** 31 Aug 2023

Abstract: In this paper, noise classification of ICF images is performed. One hundred thousand synthetic ICF images are generated to classify different kinds of noise. The noise of ICF images is neither additive nor does it follow any typical distributions. So, a deep neural network model, CNN, is designed to classify ten different noise distributions. Synthetic images are used as input to the model. With two hidden layers and an Adam optimizer, 91% accuracy is obtained. Experimental images are tested with the saved model, and the result is shown in this paper. Further study is needed to improve the accuracy of the model.

Key words: Noise, ICF, CNN, PSF, Distribution

1. Introduction

Inertial Confinement Fusion (ICF) is a method of achieving nuclear fusion by using high intensity lasers or particle beams to compress and heat a small target containing fusion fuel [13]. Double shell ICF targets consist of an ablator (outer shell), a pusher (inner shell), a mid-Z tamper (outer layer of the pusher), and a foam cushion between the shells. The inner shell is filled with a deuterium-tritium (D-T) mix. During implosion, the ablator is driven towards the pusher, compressing the fuel to fusible conditions causing asymmetries. The asymmetry in fuel compression can cause a loss in neutron yield from the fusion processes. Ablator asymmetries can arise from the asymmetric drive, presence of joint features, and presence of fill tubes. X-ray radiography provides a two-dimensional view of the object [12]. These images are hard to interpret because of the noise. For the reconstruction of images, inverse method is used. PSF (Point Spread Function) information is needed to solve the inverse problem [14]. PSF can be mathematically modelled using the concepts of Gaussian. It is usually assumed that all the noises are Gaussian. But that is not true. To reconstruct the images accurately, a good knowledge of different noise characteristics is necessary.

In this paper, noises and their varieties are given in Section 2. The method for generating synthetic data is presented in Section 3. Section 4 discusses Materials and Methods. The Result and Discussion are provided in Section 5. The Conclusion is given in Section 6.

Previous studies have discussed noise, its varieties, and its consequences. To our knowledge, no CNN

©Asia Mathematika, DOI: 10.5281/zenodo.8368998
*Correspondence: naima.naheed@benedict.edu

(Convolutional Neural Network) model to classify noise has been observed. One hundred thousand synthetic images are generated to train and test the CNN model.

2. Noise

Noise produces undesirable effects such as artifacts, unrealistic edges, unseen lines, corners, blurred objects, and disturbed background scenes. Noise is typically defined as a random variation in brightness or color information. It is frequently produced by technical limits of the image collection sensor or by improper environmental circumstances.

Noise is omnipresent: any imaging device must use a finite exposure (or integration) time, which introduces stochastic noise from the random arrival of photons. Optical imperfections and instrumentation noise result in more noise. Sampling causes noise due to the aliasing of high-frequency signal components, and digitization produces quantization errors. Further noise can be introduced by communication errors and compression [4].

In general, images can be corrupted by signal-dependent (multiplicative) noise, signal-independent (additive) noise. Let $i(\cdot)$ denote an image.The image is decomposed into a desired component, $s(\cdot)$, and a noise component, $n(\cdot)$. The most common decomposition is additive:

$$i(\cdot) = s(\cdot) + n(\cdot). \tag{1}$$

This type of noise is independent of the signal. The second most seen decomposition is multiplicative:

$$i(\cdot) = s(\cdot) * n(\cdot) \tag{2}$$

This type of noise is signal dependent. The multiplicative model can be transformed into the additive model by taking logarithms and the additive model into the multiplicative one by exponentiation [3]. For instance, (1) becomes

$$e^i = e^{s+n} = e^s * e^n$$

Similarly (2) becomes

$$\log i = \log sn = \log s + \log n$$

When coherent light strikes a surface, it is reflected. Due to the microscopic variations in the surface roughness within one pixel, the received signal is subjected to random variations in phase and amplitude. Some of these variations in phase add constructively, resulting in solid intensities, and others add in deconstructive manner, resulting in low intensities. This variation is called speckle. PSF plays a crucial role in understanding the speckle [3].

When the PSF is broad compared to the feature size of the surface roughness but small compared to the features of interest in the image, then it is concluded that the noise is exponentially distributed. Also, in this situation, the noise is often modeled as multiplicative [3, 6].

Gaussian noise is present in almost any signal. In low-light situations, image sensors must count photons. The number of photons counted is a random quantity, and the corresponding images have photon-counting noise. The grain noise in photographic films is sometimes modeled as Gaussian and sometimes as Poisson.

Many images are damaged by salt and pepper noise as if someone has sprinkled black and white dots on the image. The most frequently striking noise is additive Gaussian noise. There are important situations when neither the additive nor the multiplicative model fits the noise well [3]. Shot- Poisson and salt and pepper noise fit neither model well. In this research, ten thousand images for each noise distribution are generated. The ten different noises are given below:

1. Salt and Pepper
2. Additive Uniform
3. Multiplicative Uniform
4. Shot-Poisson
5. Multiplicative Exponential
6. Additive Exponential
7. Multiplicative Gaussian
8. Additive Gaussian
9. Multiplicative Rayleigh
10. Additive Rayleigh

3. Synthetic Data Generation

To train the convolutional neural network for the classification of noises, images paired with ground truth (label corresponding to a noise model) are needed. Due to the scarcity of experimental ICF images, synthetic radiographs are generated. This is essentially carried out in a two step process. First, a mathematical model is created. For a monochromatic X-ray source, the transmission of X-rays through a material is given by the Beer-Lambert Law,

$$T = e^{-\int_L \mu(x,y,z)\, dl}$$

where μ is the linear attenuation coefficient at position x, y, z and L is the line between the X-ray source and the object [5, 12]. These integrals are calculated using a ray tracing algorithm, provided by the python package TIGRE. Then the images are generated by applying noise through additive, multiplicative, and shot noise models with known noise distributions. For the implementation of the distributions, numpy.random API is used [1, 2].

4. Materials and Method

Input images and the corresponding outputs (ground truth), as well as initial values for the weights, are fed into the CNN model (forward method), and a measure of the error is evaluated by comparing the resulting outputs to the ground truth. The change in the error following a unit change in weights (that is, the gradient of the error with respect to the parameters) is computed using the chain rule for the derivative of a composite function (backward propagation). The parameter weights are optimized in this way. The value of the weights is then updated in the direction that leads to a decrease in the error. The procedure is repeated until the error, evaluated on unseen images, falls below an acceptable level. GPU is used here for fast computation.

The deep learning machine is a complex mathematical function mapping inputs to outputs. Python programs facilitate building deep learning projects, where PyTorch is a library. This PyTorch library emphasizes flexibility [9, 10]. This approachability and ease of use found early adopters in the research community. In the years since its first release, it has grown into one of the most prominent deep-learning tools across a broad range of applications. PyTorch provides a core data structure, the tensor, a multidimensional array that shares many similarities with NumPy arrays [7]. Numpy's random number routines produced pseudo-random numbers using combinations of BitGenerator to create sequences. A BitGenerator uses those sequences to sample from different statistical distributions [1, 2].

To classify ten different noise distributions, one hundred thousand synthetic images are generated with labels (ground truth). For this purpose, lightweight data-interchange format JSON files are created. JSON is built on image/ground truth pairs. Then they are parsed into images and labels pair. Each image has a size of 256×256 pixels.

To improve the model generalization, torchvision.transforms provide different functionalities for performing data augmentation. Two kinds of transforms are defined one is train-transform, and another is test-transform. The train-transform is composed of randomly flipping the image horizontally, rotating the image by a slight angle such as 10 degrees, resizing to 224, converting to tensors, and finally normalizing. The test-transform is like train-transform, except the rotating and flipping part are excluded. The 4-D tensors express the images. The torch.utils.data module has a class that randomly splits 70,000 of the original data images to the training set. The rest is separated into 100 images for validation and the remaining for the test set.

The main goal is also to see the decrease in training loss and validation loss. While ideally, both losses would be roughly the same value if the validation loss stays close to the training loss. The model continues to learn generalized things about the data. Overfitting in a model happens when the model's performance continues to improve on the training set but degrades on the validation set. This overfitting is usually due to the model not generalizing and memorizing the desired outputs for the training set [9]. Overfitting is avoided in the CNN model.

Figure 1. Our baseline convolutional architecture. The input images are 224 by 224 by 1 channel. Each convolutional layer is followed by pooling, next flattened, then two fully connected layers and a final dense layer with 10 outputs followed by softmax.

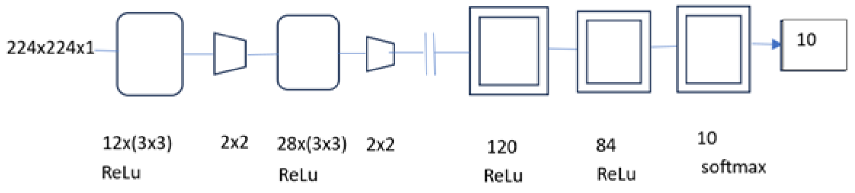

Figure 1 shows the baseline convolutional architecture of our CNN model. The first convolution layer takes the input image from 1 channel to 12 channels with kernel size 3 and stride=2, thereby giving the network a chance to generate 12 independent features that operate to discriminate low-level features of different kinds of noises. Then the nonlinear activation function ReLu is applied. Since padding is zero, two border pixels are lost for image size 224. The resulting image is pooled by the first MaxPool2d. Pooling Layer helps to down sample the images that eventually increases statistical efficiency and reduces memory requirements. The image is 111×111 pixels. At this point, the down sampled image undergoes another convolution layer that generates 28 channels of kernel size 3 and stride =2. Two border pixels are lost again. This output will consist of 28 independent higher-level features. The ReLu activation function is applied again and then pooled through another MaxPool2d. The down sampled image has size 54×54 pixels. Then the image is flattened to 6,300 pixels. The next 2 layers are fully connected layers FC1 and FC2 with outputs 120 and 84 neurons respectively. Then FC3 receives 84 neurons as input, generating 10 outputs. For a classification task, using the softmax function on the output of a network produces values that satisfy the requirements for being interpreted as probabilities. The ideal loss function for classification in this case is obtained by using the output of softmax as the input of a non-negative log likelihood function. The combination of softmax and such loss is called cross entropy in PyTorch [10].

Adam optimizer is used to update the weights in Back Propagation. Leveraging the local connectivity in CNN, 770,306 parameters are generated. After 18 epochs, it is noticed that training loss and validation loss are starting to diverge, and then it is stopped. The model is saved for future deployment. Two graphs are given below to show the performance of the model. Figure 2 gives the loss at the end of each epoch and Figure 3 gives the accuracy.

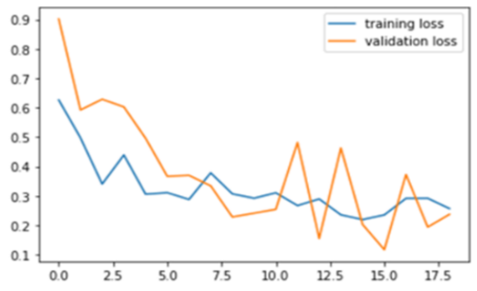

Figure 2. Loss at the end of each epoch is shown for the CNN model.

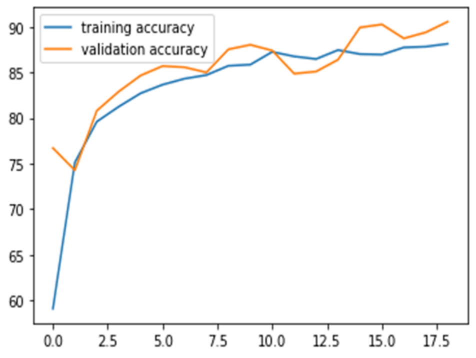

Figure 3. Accuracy at the end of each epoch is shown for the CNN model.

Finally, approximately 91% accuracy is obtained on test images.

5. Result and Discussion

To visualize the model performance, the following confusion matrix is generated.

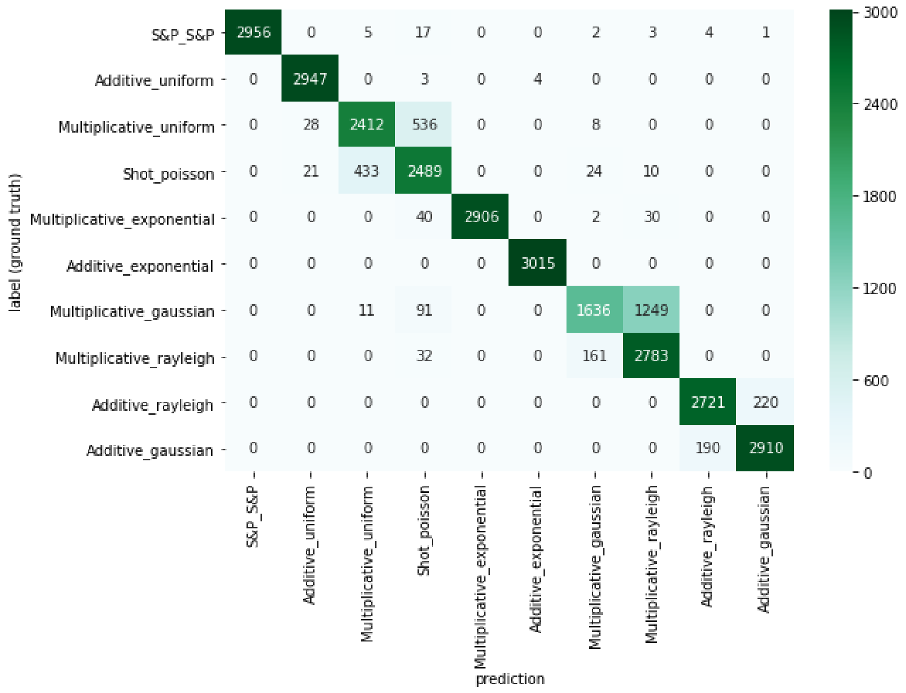

Figure 4. Confusion matrix is generated from the CNN model.

From the Figure 4, it is noted that the largest error occurs when the truth label of the image is multiplicative Gaussian noise, but it is predicted as multiplicative Rayleigh noise. The Rayleigh distribution is related to the Gaussian distribution through the property that given two independent normally distributed random variables $X \backsim N(0, \sigma^2)$ and $Y \backsim N(0, \sigma^2)$, and then the variable $R = \sqrt{X^2 + Y^2}$ is a Rayleigh distributed random variable with parameter σ. So, it is apparent that Rayliegh and Gaussian are closely related. As a result, the CNN model performed poorly when distinguishing between multiplicative Gaussian and multiplicative Rayleigh noises. Second and third source of error comes when the model predicts Shot -Poisson but it is Multiplicative Uniform and vice versa respectively. It is known that the quantum effects of photons arriving stochastically create Shot Poisson noise [8]. When the parameter λ is large, the Poisson distribution is well approximated by the Gaussian with mean and variance both equal to λ. Multiplicative Uniform noise is the opposite of the heavy tailed noise, whose tails are approximately zero [11]. For this reason, CNN model diagnosed Shot -Poisson as Multiplicative Uniform noise and Multiplicative Uniform as Shot - Poisson. It is noticed that the CNN model performed very good in the case of additive exponential case and in additive uniform. No misdiagnosis happened.

Experimental images are run through the model. One of the experimental images is given below in Figure 5.

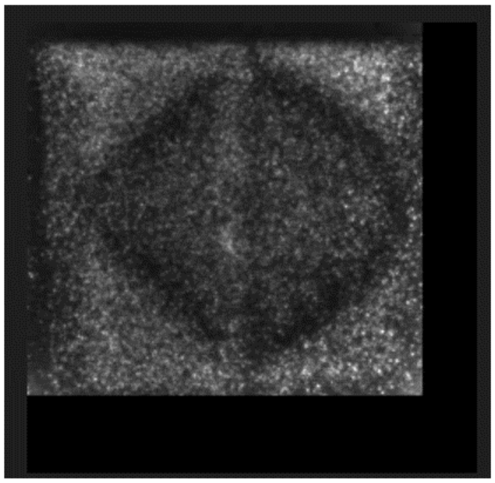

Figure 5. An experimental image with unknown noise.

The CNN model classified the noise as Shot-Poisson. It is beyond human capabilities to identify the noise by observing the image. Our CNN model achieved the task with 91% accuracy.

6. Conclusion

This project uses a deep neural network model, CNN, to classify ten different noise distributions. Synthetic images are used as input to the model. With two hidden layers and an Adam optimizer in the model, 91% accuracy is obtained. In the future, different deep-learning models will be used to classify noise. By taking the means, the prediction can be improved. The ensemble prediction will be more vigorous if each model is skillful but in different ways.

Acknowledgment

The first author is supported by the DoE VFP program and DoE/NNSA OFE program (ICF). The experiment was performed in Los Alamos National Lab. **Z.W. gave the idea, B.W. and N.N. built the model and ran the experiments, N.N. interpreted the results, N.N. wrote the paper. All authors read and approved the final submitted version.**

References

[1] "Random sampling (numpy.random) — NumPy v1.24 Manual," numpy.org. https://numpy.org/doc/stable/reference/random/index.html

[2] "Random Generator — NumPy v1.24 Manual," numpy.org. https://numpy.org/doc/stable/reference/random/generator.html#

[3] C. Boncelet. Image Noise Models. In: Handbook of Image and Video Processing, 2005, pp. 143–167.

[4] S. Chavan and Dr. N. S. Choubey. Review on Various Noise Models and Image Restoration Techniques. In: International Journal of Development Research, vol. 7, no. 9, Sep. 2017, pp. 15048–15053.

[5] M. Falato, B. Wolfe, N. T. T. Nguyen, X. Zhang, and Z. Wang, Contour extraction of ICF images Without Ground Truth. https://doi.org/10.48550/arXiv.2211.04597.

[6] S. J. Fletcher. Univariate Distribution Theory, In: Data Assimilation for the Geosciences, doi: 10.1016/B978-0-12-804444-5.00003-9.

[7] A. S. Glassner. Deep learning : a visual approach. San Francisco, Ca: No Starch Press, 2021, pp. 429–493.

[8] B. C. Levy. Poisson Process and Shot Noise. In: Random Processes with Applications to Circuits and Communications, Cham: Springer, 2019, pp. 235–258.

[9] G. Antiga, E. Stevens, and T. Viehman. Deep Learning with PyTorch. Simon and Schuster, 2020, pp. 39–68.

[10] V. Subramanian. Deep Learning with PyTorch. Packt Publishing Ltd, 2018, pp. 37–78.

[11] H. M. Taylor and S. Karlin. An Introduction to Stochastic Modeling. Academic Press, 1998, pp. 267-318.

[12] B. T. Wolfe et al. Machine Learning for Detection of 3D Features using sparse X-ray data. https://doi.org/10.48550/arXiv.2206.02564.

[13] R. Betti and O. A. Hurricane. Inertial-confinement fusion with lasers. In: Nature Physics, vol. 12, May 2016, pp. 435–448.

[14] S. Gazzola, P. C. Hansen, J. G. Nagy. IR Tools: A MATLAB Package of Iterative Regularization Methods and Large-Scale Test Problems. 2017. doi:10.48550/ARXIV.1712.05602.